**블록체인** 발행 주체(중앙 은행 / 채굴 참여자) 도입 기관(국가 / 개발자 그룹(사토시 나카모토)) 가치 보증 (국가 기관 및 은행 / 소프트웨어 알고리즘) 화폐 소유 관리(실물 현금, 은행 잔액 기반 / 공개 장부, 거래 내역 기반) 공급 결정권(국가 기관 및 은행 / 발생양 및 총량이 결정됨)

**Peer-to-peer 자원 공유**, 모든 거래내역 공유, **PKI매커니즘** 개인키생성(256비트의 무작위 숫자 생성 약 1077, 개인키로

**해시코드알고리즘** 입력값을 256bit 고유출력값으로(출력값으로 역산 못함) **비트코인구조** Block header• 이전 블록의 해시코드• (Merkle Root)전체 트랜잭션의 해시코드 • (Timestamp)블록 생성 시간• (Difficulty)문제 난이도• (Nonce)문제 정답, 문제 예(해시코드를 0000으로 시작하는 추가숫자를 찾아라) / 2140년까지, 2천1백만 코인 채굴가능

**Survey**

**Questionnaire**

**Customer**

**Developer**

**Use Case**

**Brain Storming**

**Requirements Workshop**

**Prototyping**

**Interview**

**Role Playing**

**불법거래방지** 가장긴쪽에(6승인이 되면 거래승인으로 간주), 먼저 들어온 쪽에 블록을 붙임.(단, 51% 위협 존재, 집중도 0.33)

**이더리움** 스마트계약 가능, **문서무결성검증**(문서ID, 문서의 해쉬코드), 4차 산업혁명, 9번째 프로토콜

**현재한계점** • Bitcoin vs. Visa • Throughput : 7tps to 5,000tps • Latency : 10 min. to 2 sec. • Size and bandwidth : 50 GB to 214PB/y • Security : 51% attack • Wasted resource : $15 million/d • Usability : developer-friendly API • Versioning, hard forks, multiple chains /• AML(anti money laundering) vs. Privacy /• Less node, Less trust /• Trust vs. Validation delay

**블록체인 장점** 1.cost 대비 보안성이 좋다. 2.탈중앙화(직거래 서비스) 3.Smart contract 4.신뢰도 있는 시스템

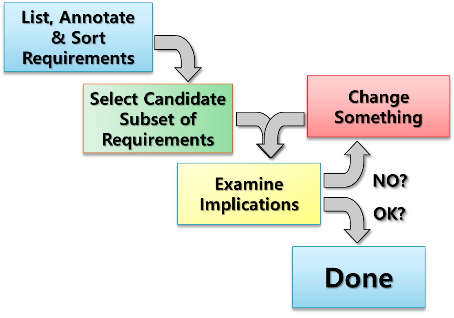
**ch1. Requirments?** 실생활영역의 문제와 개발영역의 솔루션, 유저가 문제해결 또는 목표달성을 위해 조건 및 기능을 필요, system what **Repair Cost** Requirements단계(1~2) Design(5) Coding(10) Unit test(20) System test(50, 60%가 이단계에서) Maintenance(200) **Three Level of SWR** 기획(vision, goals), 요구사항추출 및 걸러내기(triage), 일반적 요구사항, 모델링(use cases), 기능적요구사항, SW Requirements Specification(SRS)

**ch2. 요구사항추출(Requirements Elicitation)** •The Art of Sending Appropriate Stimuli to Stakeholders so that the responses are worth listening to •The Art of Establishing an Environment in which Stakeholders are willing and able to describe their problems and needs **Things to Remember when Elicitating Requirements** 1. Don’t Lose Sight of the Goal(목적 : 엉뚱한 시스템을 만들지 않도록, 고객들도 초기단계부터 관련시켜 이후 소통을 원활) 2. Care(개발자가 먼저 열정을 보여야) 3. Think Who’s Smart(고객이 스마트하다고 느끼도록) 4. Make Things Easy to Understand(모호하지 않도록 명확히) 5. One Stakeholder Can’t Speak for All(user, customer, marketing, subject matter experts, developers, development managers, testers, loser users : 이시스템으로 피해가 예상되는 그룹의 의견) 6. 적합한 추출기법 사용 •기법(Interviews, Role Playing, Brainstorming, Requirements Workshop, Prototyping, Survey/Questionnaire, Use Case, •A single method may not be sufficient •Consider requirements’ size, complexity, etc) 7. Accept Requirements Changes 8. Manage Elicitated Requirements(requirements의 근거, 이유를 기록, 관리) **Elicitation Methods** 1. Interview(before, during, after) 2. Braining Storming 3. Observation(tacit) 4. Presentation-Based Approaches

**Post-Elicitation** 1)lists, Maintaining a list of requirements can support all activities of requirements 2)요구사항추출의 결과를 볼 수 있도록(list화, old/new candidate)

**Ch3 요구사항분석** 요구사항의 구체화 1. Requirements Modeling •To Understand the requirements - Completeness, Consistency(앞뒤), Testability(전체흐름) •Principles of requirements modeling - Early artifacts, Cheap to make, Easy to visualize and optimize - Abstraction: 실제를 단순화 시켜서 표현, Partitioning, Projection 2. Requirements Prioritization 3. Requirements Triage 4. Agreeing on Requirements **Modeling Methods** •Structured Analysis(구조적 분석법 :Data Flow Diagram (DFD), Entity-Relation Diagram (ERD), State Transition Diagram (STD)) •Use Case Analysis (Use Case Modeling (UC)) •Goal and Scenario Based Analysis (Goal-Scenario Modeling (GS)) **구조적분석** •데이터/기능/상태를 중심으로 개념을 분석하는 방법

•모델링 기법 Data/Function-Oriented Modeling(하향식 세분화, 그래픽 중심의 분석 – DFD(Data Flow Diagram), ERD(Entity Relation Diagram)) State-Oriented Modeling(기능 순서 분석 : 연속적으로 발생하는 입력에 대한 함수에 적합, FSM(Finite State Machine) 이용 – STD(State Transition Diagram)) **DFD** 필요한 데이터를 간략히 보여준다. 업무흐름 중심의 어플리케이션을 위한 요구사항으로 유용하다. **E-R Modeling** 시스템 내부와 외부의 데이터를 그래픽 형태로 표현하는 모델(전문가에게는 유용하나 사용자에게는 어려움, 데이터베이스 설계에 유용, DFD의 자료저장소가 가지는 개체간의 관계 및 속성을 나타냄, 자료중심의 어플리케이션을 위한 요구사항으로 사용 가능) **STD** 입력/조건 변화에 따른 상태변화를 표현하는 모델, 빠뜨린 기능, 불명료한 기능을 찾는데 유용 **Use Case Analysis 사용자 중심 요구사항 모델링** • Use Case와 Actor간 상호작용을 통해 시스템 분석(사용자 비즈니스와 애플리케이션 영역을 이해, 작업 중심적: Diagram으로 요구사항 이해) •Use Case는 각각 독립적(High priority Use Case = High priority requirements, 단순한 Use Case 나열: 전체적인 시스템 요구사항 이해 어려움, 비기능적인 요구사항은 표현이 어려움) •표준 Graphic 표기(Actor와 system의 관계, Communication 개선, 해석의 Risk 감소) **Goal & Scenario Modeling** •Inputs: Initial requirements •Outputs: Goal tree •Abstraction levels provide: Separation of concern, Levels of goal & scenario modeling, The four levels are: Business, Service, Interaction(고객과), and Internal(시스템 내부)

**Requirements Prioritization** factors that affects priority(customer want, cost, time, technologically, organizationally, business benefit) **Requirements Prioritization Methods** $100 Test(“Prisoners Dilemma” voting; collusion), YES/NO Vote(But, what does “no” mean?), -2 to +2(Sometimes it’s important to record all votes independently), AHP(Analytical Hierarchy Process)(step - List all features, relative benefit, relative penalty:미포함시, relative cost, relative degree of technical or other risk, Calculate a priority number, Sort) **Service Requirements Triage** Tips for requirements triage •Maintain requirements in lists •Annotate requirements, by at least relative priority and cost-to-satisfy •Don’t ignore triage •Establish practices that pit the team “against” the problem, rather than between team members •Let schedule drive requirements inclusion •Involve representatives from all key groups(C,D,Financin •Follow a sensible triage process

**Ch5 What is Use Case Modeling?** •A means for capturing the desired behavior for the system under development•A way to communicate the system's behavior •Identifies who or what interacts with the system and what the system should do •A way to verify all requirements are captured •A planning instrument

**Benefits of Use Cases** •Give context for requirements •Are easy to understand •Facilitate agreement with customers •Illustrate why the system is needed(Use cases: why the system is used, Actors: who/what wants to interact with the system) The idea behind use cases is to decide what the system will be used for before defining what the system is supposed to do. **Actor** 개발한 시스템이 인터페이스해야 하는 것들, **Use Case** 시스템이 actor에게 제공하는 주요한 서비스, S/W 요구사항을 포함(엘리먼트 상호간 독립성이 있어야하며 상호연관성이 많으면 묶음, 내부 spec.은 응집력이 있어야함, 엘리먼트간 밸런스가 맞아야) **Use Case Tips** •Describe only the events visible to the actor(What the actor does, What the system does in response) •Make use cases provide value to an actor •Detail until everyone has a common understanding of the requirements, then stop •Sketch the user interface, but don’t detail it •Use agreed-upon terms and vocabulary •Use precise language. **Steps for Creating a Use Case Model** 1. Find actors and use cases(Identify and describe actors, Identify and describe use cases) 2. Write the use cases(Outline all use cases, Prioritize and detail the use cases)

**Checkpoint for Use Cases** •The use-case model clearly presents the behavior of the system; it is easy to understand what the system does by reviewing the model •All use cases have been identified; the use cases collectively account for all required behavior. •All functional requirements are mapped to at least one use case. •The use-case model contains no superfluous behavior; all use cases can be justified by tracing  them back to a functional requirement. •Do the use cases have unique, intuitive, and explanatory names so that they cannot be mixed up at a later stage? If not, change their names. •Do customers and users alike understand the names and descriptions of the use cases? •Does the brief description give a true picture of the use case? •Is each use case involved with at least one actor? •Do any use cases have very similar behaviors or flows of events? **Checkpoint for Actors** •Have you found all the actors? That is, have you accounted for and modeled all roles in the system's environment? •Is each actor involved with at least one use case? •Can you name at least two people who would be able to perform as a particular actor? •Do any actors play similar roles in relation to the system? If so, you should merge them into a single actor.

**Ch6 An Outline of Specifying SRS Using Use Case** Use case driven requirements specification is composed of as follows: 1. Create Use Case Specification 2. Create Non-functional Requirements Specification 3. Organize SRS using UC Spec and Supplementary Spec

**Use Case 순서** 1.1. Identify actors and use cases 1.2. Outline each use case(Brief Description, Basic Flow of Events, Identify Alternative Flows of Events 1.3. Detail each use case (Add detail : Pre and post-conditions, special requirements, relationships, use-case diagrams, and so on 1.4. Structure each use case

**Detail each use case** <Use-Case Name>\*1. Brief Description\*2. Flow of Events\*(Basic Flow of Events, Alternative Flows of Events) 3. Special Requirements 4. Preconditions 5. Postconditions 6. Extension Points 7. Relationships\* 8. Use-Case Diagrams 9. Other Diagrams/Enclosures

**Use Case Checkpoints** •Are the actor interactions and exchanged information clear? •Does the communication sequence between actor and use case conform to the user's expectations? •Is it clear how and when the use case's flow of events starts and ends? •Is the subflow in a use case modeled accurately? **Base --> include** 반드시 포함 **Base <-- extend** 조건부포함 **SRS Package** use case model + supplementary specifications